Real-Time Scheduling Analysis

Dr. Joseph Leung, Department of Computer Science
New Jersey Institute of Technology, Newark, NJ 07102

Mr. Marvin Stewart, Mr. Charles Bergen
WOW IS, 900 Briggs Road, Mt. Laurel, NJ 08054

1. Introduction

Computers have been used to control real-time tasks that are commonly found in the avionics industry. Typically, these real-time tasks are periodic, with strict deadline constraints to meet. A real-time task is characterized by three parameters, e, d and p, where e represents its maximum execution time, d represents the relative deadline that it must meet, and p represents the period of the task. We assume that e is less than or equal to d which in turn is less than or equal to p. In this setting, the task makes an initial request at time 0, and thereafter at times kp, where k is a positive integer. The kth request (which occurs at time (k-1)p) must complete at time (k-1)p+d, which is the deadline for the request. During the time interval between (k-1)p and (k-1)p+d, the real-time task requires e units of processing time. If several real-time tasks share a limited number of computational resources (such as CPU, I/O devices, etc.), how do we schedule them so as to ensure that all deadlines of all requests of all real-time tasks are met?

Prior to 1970, this scheduling problem was handled by handcrafting a schedule with length equal to the super cycle of the real-time tasks, where the super cycle is the least common multiple of the periods of the real-time tasks. At run time, CPU time is allocated to the real-time tasks according to the handcrafted schedule. There are two problems with this approach. First, it is difficult to handcraft a schedule, since the super cycle is rather large. Second, it is not easy to change the schedule when there are changes in the parameters of the tasks. Any changes in the parameters require a new schedule be handcrafted again. Thus, this approach is time-consuming and error prone.

Since the early 1970, computer scientists have tried to devise better methods to handle this scheduling problem. A scheduling algorithm, known as the Earliest Deadline First (EDF) rule, was proposed to schedule the real-time tasks. The EDF rule schedules, at each moment of time, the task whose deadline is closest to the current time. Ties can be resolved by an arbitrary tie-breaking rule. The EDF rule has been shown to be optimal for one processor, in the sense that it can schedule any set of real-time tasks for which a feasible schedule exists. The EDF rule is a great improvement over the previous approach of handcrafting a schedule, since it can be automated. Furthermore, we can construct a schedule with length equal to the super cycle and check if all the deadlines are met. A computer program can automate all the schedule construction and the deadline verification.

The EDF rule, however, has a drawback in that it incurs significant overhead in the scheduling of tasks. Moreover, it can only be implemented by software. Because of this deficiency, computer scientists looked into the possibility of scheduling rules, which can be implemented by hardware alone. A class of scheduling algorithms, known as Fixed-Priority algorithms, has been proposed. The idea of Fixed-Priority algorithms is to assign a priority to each real-time task at the outset. At run time, a task at a higher priority will always receive the CPU before a task at a lower priority. The priority of a task can be attached to the priority of the interrupt system of the hardware. In this way, the scheduling algorithm can be implemented by the hardware alone.

The key issue for a Fixed-Priority algorithm is the method priorities are assigned to the tasks. Liu and Layland [1] proposed the Rate-Monotonic algorithm, which assigns priorities to the tasks in ascending order of their periods; i.e., the task with the smallest period is assigned the highest priority and the task with the largest period is assigned the lowest priority. Liu and Layland [1] showed that Rate-Monotonic is optimal for the special case

where the deadline of each task is identical to its period. Leung and Whitehead [2] later showed that the Deadline-Monotonic algorithm is optimal for the general case where the deadline of each task needs not be identical to its period. The Deadline-Monotonic algorithm assigns priorities to the tasks in ascending order of their deadlines; i.e., the task with the smallest deadline is assigned the highest priority and the task with the largest deadline is assigned the lowest priority.

2. Project Description

There are two main objectives in this project. First, we will carry out an industry survey for the need of real-time scheduling analysis and the common practice in dealing with this issue. Our industrial partner, WOW IS, will carry out this task. The second objective is to further extend the theory developed by Liu and Layland as well as Leung and Whitehead to incorporate practical constraints that were not included in their studies. Dr. Joseph Leung and a PhD student (Hairong Zhao) at NJIT will carry out this task.

There are four theoretical issues to study in this project:

(A) Limited Priority Level -- All computer systems have a limited number of priority levels in their interrupt systems. The theory developed in [1,2]assumes that each real-time task has a distinct priority. When the number of tasks exceeds the number of priority levels, we need to map several tasks into the same priority. When several tasks at the same priority level make a request, the order of execution is arbitrary. In this setting, how do we assign priorities to the real-time tasks and how do we check if all deadlines will be met under any circumstances?

(B) Multiprocessors -- How do we assign real-time tasks to a minimum number of processors? It is important to use the fewest number of processors since CPU can consume energy, which is a scarce resource in an aircraft.

(C) Fault-Tolerant -- The models studied in [1,2] assume that there is no time loss due to faults. In practice, faults can occur at the hardware or software level. How do we ensure that all (or most) deadlines are met in the face of time loss due to faults? We propose that each task has two versions of implementation, the fast one and the slow one. The slow one takes more time but produce a high quality result, while the fast one takes less time but produce a less quality result. The slow version will be used in normal times. When system faults occur, we immediately switch to the fast version. In this setting, how do we quantify the number of missed deadlines?

(D) I/O and CPU Scheduling -- The models studied in [1,2] assume dedicated I/O devices for each real-time task. We want to extend the model to include shared I/O devices and study the CPU and I/O scheduling problem.

3. References

[1] C.L. Liu and J.W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, J. ACM 20 (1973), 46-61.

[2] J.Y-T. Leung and J. Whitehead, On the complexity of fixed-priority scheduling of periodic, real-time tasks, Performance Evaluation 2 (1982), 237-250.